University of Pittsburgh

Symbolic Solution for Linear Systems

Week 7

MEMS 1140—Introduction to Programming in Mechanical Engineering

Learning Objectives (L.O.)

At the end of this lecture, you should understand/be able to:

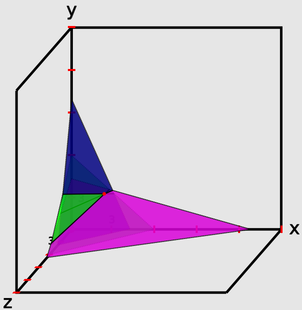
- ❑ Plot a linear system using symbolic equations;
- ❑ Write a linear system using the Symbolic Toolbox;
- ❑ Solve the linear system using the Symbolic Toolbox;
- ❑ Access the solution values as double-precision floats.

Table of Contents (ToC)

1 – Recall Previous Lecture

We used the following example in Lecture 5.1:

$$\begin{cases} 2x + y + 3z = 10 \\ x + y + z = 6 \\ x + 3y + 2z = 13 \end{cases}$$



This graph was hand-drawn and not to scale!

Let's use the Symbolic Toolbox to replicate this plot.

1 – Define Symbolic Variables

Note that the $\mathbf{y}$ axis is oriented upwards in the previous graph.

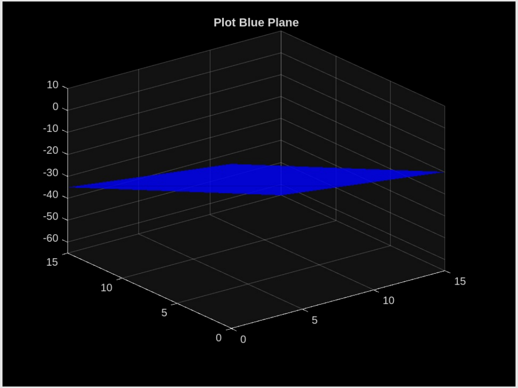
Define symbolic variables to represent each equation in the form $y(x, z)$:

$$\begin{cases} y_1(x,z) = (10 - 2x - 3z) \\ y_2(x,z) = (6 - x - z) \\ y_3(x,z) = \dfrac{(13 - x - 2z)}{3} \end{cases}$$

```
syms y1(x,z) y2(x,z) y3(x,z)
```

1 – Plotting — Plot Blue Plane


```
y1(x,z) = (10 - 2*x - 3*z);
fsurf(y1(x,z), [0 15 0 15], ...
   'FaceColor', 'b', ...
   'FaceAlpha', 0.8, ...
   'EdgeColor', 'none')
```



The **fsurf** command plots a symbolic function **f(u,v)** over the interval:

[umin umax vmin vmax].

1 – Plotting — Set Vertical Limits

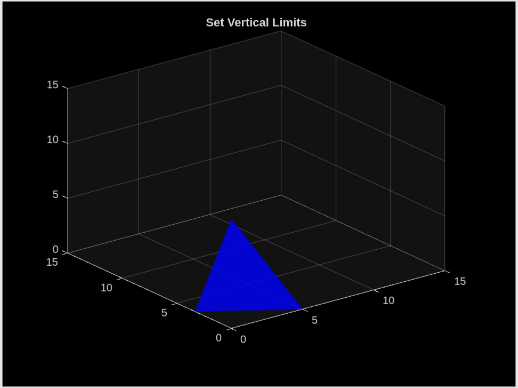

```
zlim([0,15])
```

Note that the original plot only considers positive **`x`**, **`y`**, **`z`**.

The interval **`[0 15 0 15]`** specifies this for **`x`** and **`z`**.

`zlim([0,15])` sets the **`y`**-axis bounds.

*Recall that **`y`** is oriented along MATLAB's **`z`**.*

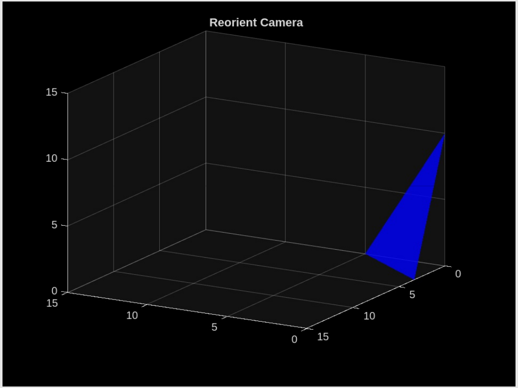


1 – Plotting — Reorient Camera


```
view(-150,20)
```

Now we adjust the camera position to closely match the original graph.

`view(az,el)` takes azimuth and elevation values as arguments to orient the camera around the plot box.



1 – Plotting — Flip X Axis

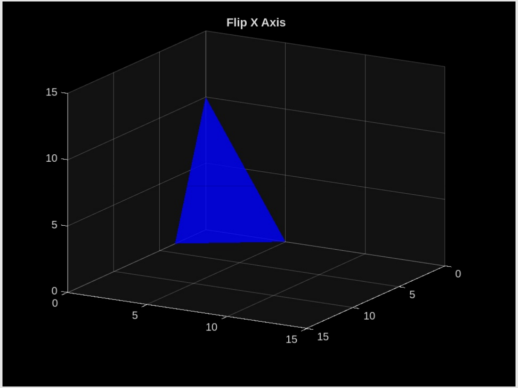

```
set(gca, 'XDir', 'reverse')
```

Note that the **x** axis was backwards before.

It read 15 → 0 from left to right.

This command flips the direction of the **x** axis.

It now reads 0 → 15 from left to right.



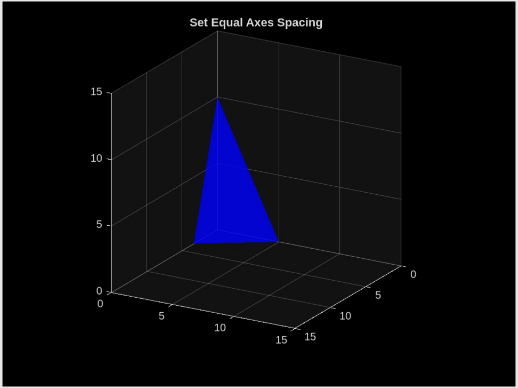
1 – Plotting — Set Equal Axes Spacing

➩ L.O.1
❏ L.O.2
❏ L.O.3
❏ L.O.4

```
daspect([1,1,1])
```

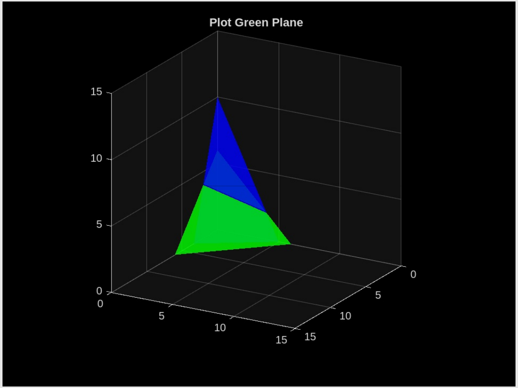
`daspect([x,y,z])` scales the axes according to the ratio of the argument.

We want a uniform scaling for this example!



1 – Plotting — Plot Green Plane


```
hold on
y2(x,z) = 6 - x - z;
fsurf(y2(x,z), [0 15 0 15], ...
   'FaceColor', 'g', ...
   'FaceAlpha', 0.8, ...
   'EdgeColor', 'none')
```

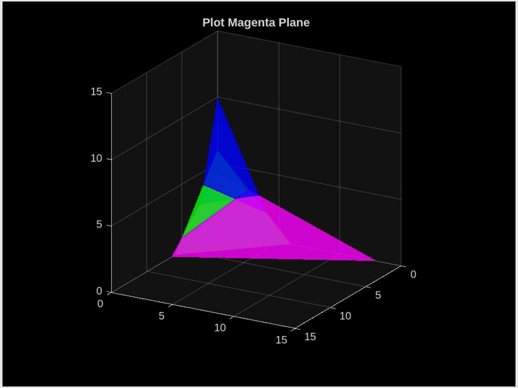


Note that the first command **hold on** preserves the contents of the figure before drawing the green plane.

1 – Plotting — Plot Magenta Plane


```
y3(x,z) = (13 - x - 2*z)/3;
fsurf(y3(x,z), [0 15 0 15], ...
   'FaceColor', 'm', ...
   'FaceAlpha', 0.8, ...
   'EdgeColor', 'none')
```

Add the last plane.



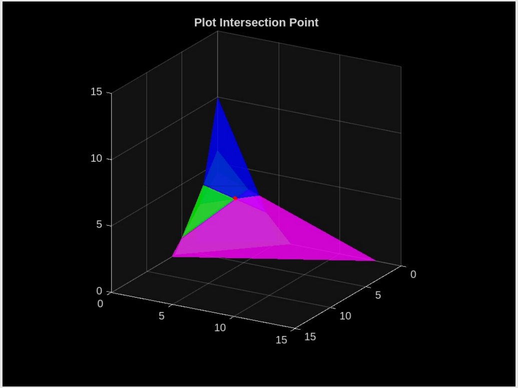
1 – Plotting — Plot Intersection Point

⇨ L.O.1
❏ L.O.2
❏ L.O.3
❏ L.O.4

```
scatter3(2,1,3,20,'red','filled')
```

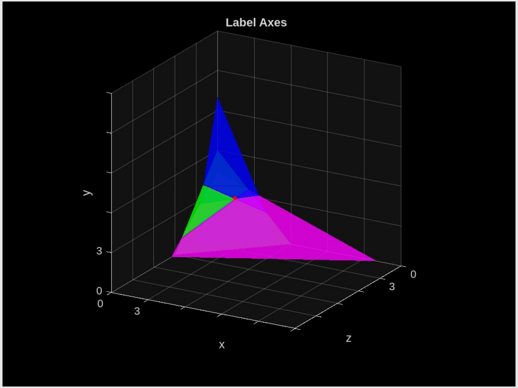
`scatter3(x,y,z)` plots a point in the figure at the provided coordinates.

The last three arguments specify the size `20`, the color `red`, and to fill the circle.



1 – Plotting — Label Axes


```
xlabel('x')
ylabel('z')
zlabel('y')
xticks(0:3:15)
xticklabels([0,3])
yticks(0:3:15)
yticklabels([0,3])
zticks(0:3:15)
zticklabels([0,3])
```

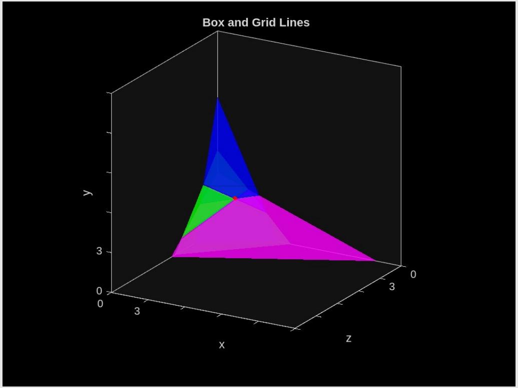


These replicate the ticks and labels from the original.

1 – Plotting — Box and Grid Lines


```
grid off
box on
```

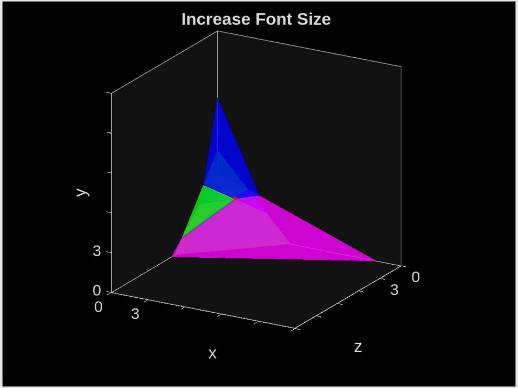
To get a more clean plot box, these commands turn off the grid lines and turns on the background box edges.



1 – Plotting — Increase Font Size


```
set(gca, 'FontSize', 16)
```

And finally, increase the font size for better legibility.



2 – Write a Symbolic Linear System

The Symbolic Toolbox offers a command to solve systems of equations: `solve(eqns,vars)`.

`eqns` is a vector of symbolic equations and `vars` is a vector of the symbolic variables in those equations.

This can be illustrated with a simple one-equation system:

$$\begin{cases} 5x + 7 = 12 \end{cases}$$

2 – Write the Symbolic Equation

First, define the symbolic variable **x**:

```
>> syms x
```

Command Window

Then, the equation is defined as follows:

```
>> equation = 5*x + 7 == 12;
```

Command Window

Note the double == used for equality within the equation!

2 – Solve the Symbolic Equation

Now the **`solve`** command can be used:

```
>> [x_solution] = solve(equation, x)     % (12 - 7) / 5 = 1
x_solution =
1
```

Command Window

This can be used to solve complicated individual equations.

But it can also be used to solve systems of multiple equations!

2 – System of Multiple Equations

Let's return again to our example with three equations:

$$\begin{cases} 2x + y + 3z = 10 \\ x + y + z = 6 \\ x + 3y + 2z = 13 \end{cases}$$

2 – System of Multiple Equations

Let's return again to our example with three equations:

$$\begin{cases} 2x + y + 3z = 10 \\ x + y + z = 6 \\ x + 3y + 2z = 13 \end{cases}$$

The first equation is written symbolically as follows:

```
syms x y z
equation_1 = 2*x + y + 3*z == 10;
```

2 – System of Multiple Equations

Let's return again to our example with three equations:

$$\begin{cases} 2x + y + 3z = 10 \\ x + y + z = 6 \\ x + 3y + 2z = 13 \end{cases}$$

The second equation is written symbolically as follows:

```
syms x y z
equation_2 = x + y + z == 6;
```

2 – System of Multiple Equations

Let's return again to our example with three equations:

$$\begin{cases} 2x + y + 3z = 10 \\ x + y + z = 6 \\ x + 3y + 2z = 13 \end{cases}$$

The third equation is written symbolically as follows:

```
syms x y z
equation_3 = x + 3*y + 2*z == 13;
```

3 – Solve a Linear System

All together, the system of equations is written as follows:

```
>> syms x y z
>> equation_1 = 2*x + y + 3*z == 10;
>> equation_2 = x + y + z == 6;
>> equation_3 = x + 3*y + 2*z == 13;
>> eqns = [equation_1, equation_2, equation_3];
>> vars = [x, y, z];
```

Command Window

Note that all 3 equations and all 3 variables have to be stored in corresponding vectors!

3 – System Solution

The system is solved as follows:

```
>> solution = solve(eqns, vars)
solution =

  struct with fields:

    x: 2
    y: 3
    z: 1
```

Command Window

This is the same result as in the previous lecture!

4 – What is a Structure Array

Note that the **solution** variable is denoted as a **struct**.

In MATLAB, structure arrays are used to store different information together.

For example, a **struct** that describes this course might be defined as follows:

```
course.name = 'Introduction to Programming in Mechanical Engineering';
course.department = 'MEMS';
course.number = 1140;
```

4 – Data Storage in a `struct`

MATLAB outputs this example **`struct`** as follows:

```
>> course
course =

  struct with fields:

          name: 'Introduction to Programming for Engineers'
    department: 'MEMS'
        number: 1140
```

Command Window

4 – Accessing Elements in a `struct`

Dot notation is used to access specific elements of a `struct`.

Accessing the name:

```
>> course.name
ans =
    'Introduction to Programming for Engineers'
```

Command Window

Note that the *character array itself* is returned by this call.

4 – Accessing Elements in a `struct`

Dot notation is used to access specific elements of a `struct`.

Accessing the home department:

```
>> course.department
ans =
    'MEMS'
```

Command Window

Note that the *character array itself* is returned by this call.

4 – Accessing Elements in a `struct`

✓ L.O.1
✓ L.O.2
✓ L.O.3
⇨ L.O.4

Dot notation is used to access specific elements of a `struct`.

Accessing the course number:

```
>> course.number
ans =
     1140
```

Command Window

Note that the *numeric value itself* is returned by this call.

4 – Linear System Example

✓ L.O.1
✓ L.O.2
✓ L.O.3
➪ L.O.4

Let's return to the linear system from earlier.

To access the values of each component, use dot notation with each variable:

Accessing **x**:

```
>> solution.x
ans =
2
```

Command Window

4 – Linear System Example

Let's return to the linear system from earlier.

To access the values of each component, use dot notation with each variable:

Accessing **`y`**:

```
>> solution.y
ans =
3
```

Command Window

4 – Linear System Example

Let's return to the linear system from earlier.

To access the values of each component, use dot notation with each variable:

Accessing `z`:

```
>> solution.z
ans =
1
```

Command Window

4 – Non-Integer Symbols

✓ L.O.1
✓ L.O.2
✓ L.O.3
⇨ L.O.4

Results are stored in MATLAB as **`syms`** (symbols), not the normal `double` class.

If the values are not integers, they will be reported as fractions:

```
solution =
  struct with fields:
    x: 11/5
    y: 16/5
    z: 3/5
```

Command Window Output

4 – Converting Symbols to Doubles

The `double(x)` command can be used to convert `x` into the floating-point format.

In this example:

```
>> x_double = double(solution.x)
x_double =
     2.2000
```

Command Window

The symbol `x` (`11/5`) is converted into a double (`2.2`).

5 – Summary

✓ L.O.1
✓ L.O.2
✓ L.O.3
✓ L.O.4

This lecture covered:

✓ How to plot a linear system using symbolic equations

Each 3D equation is written as a symbolic equation **`f(u,v)`** and plotted using **`fsurf`** over a given domain in **`u`** and **`v`**.

5 – Summary

✓ L.O.1
✓ L.O.2
✓ L.O.3
✓ L.O.4

✓ How to write a linear system using the Symbolic Toolbox

With symbolic variables, the equations are written symbolically, just as they look in math. Use `==` for the equality *within* the equation.

✓ How to solve a linear system using the Symbolic Toolbox

All equation variables are stored an `eqns` array, and the variables are stored in a `vars` array. The `solve` command is then use to automatically solve the system of equations.

5 – Summary

✓ L.O.1
✓ L.O.2
✓ L.O.3
✓ L.O.4

✓ How to access the solution values as double-precision floats

The return value from the **`solve`** command is a structure array. The elements of the **`struct`** can be accessed using dot notation and then converted to double-precision floats using the **`double`** command.